Outline

Today: Continue with discussion of the two views of the function of early visual coding

- Efficient coding
  - Task neutral
    - 1st order statistics
      - point-wise non-linearities, histogram equalization, receptors and information capacity
    - 2nd order statistics
      - lateral inhibition, ganglion cells and predictive coding
      - opponent color processing (principal components analysis)
      - cortical representation
        - Decorrelation, PCA
  - 3rd order?

- Local image measurements which correlate well with useful surface properties
  - task specific--find "significant" intensity changes
  - edge detection
    - 1st and 2nd spatial derivatives
    - relation to center-surround and oriented receptive fields

Initialize

- Read in Statistical Add-in packages:

  Off[General::spell1];
  << Statistics`DescriptiveStatistics`
  << Statistics`DataManipulation`
  << Statistics`NormalDistribution`
  << Statistics`ContinuousDistributions`
  << Statistics`MultinormalDistribution`

1) Define 64x64 pixel cup image (cup), or

- Evaluate this: cup

  width = Dimensions[ cup ][[1]];  

2) alternatively import image file

  cup = Import[ Experimental`FileBrowse[False] ];
  cup = cup /. Graphics -> List;
  cup = cup[[1, 1]];
  width = Dimensions[ cup ][[1]];
Problems with edge detection

2nd order statistics & Principal components analysis (PCA)

Introduction to PCA

Principal components analysis (PCA) is a statistical technique that is applied to an ensemble of n-dimensional measurements (vectors or in our case images). To do PCA, all one needs is the autocovariance matrix and a good PCA algorithm. Good because images are big enough (p=mxn), and the covariance is much bigger (p^2).

PCA finds a matrix that transforms the input vectors into output vectors, such that output elements are no longer correlated with each other. There is more than one matrix that will do this however, and PCA find the matrix which is a rigid rotation of the original coordinate axes, so it preserves orthogonality. (The Fourier transform is also a rotation.) Further, the new coordinates can be ordered in terms of variance. The new coordinates turn out to be eigenvectors of the covariance matrix. The directions or eigenvectors with the biggest variances are called the principal components. So the dominant principal component has the most variance, and so forth. For data that are highly redundant, PCA can be used to eliminate dimensions that account for little of the total variance--i.e. those with small eigenvalues.

PCA is important in computational models of visual processing (See Wandell, pages 254-258). For example, PCA has been used to account for and model:

- opponent color processing
- visual cortical cell development
- efficient representation of human faces
- face recognition given variability over illumination

There has been considerable growth in the area of theoretical neural networks and PCA. An introduction to some of the ideas is given in the optional section below.

Standard computer statistical packages provide the tools for doing PCA on large data sets. Below we try to provide intuition and background into the computation of principal components.

Statistical model of a two-variable input ensemble

Consider a two variable system whose inputs are correlated. The random variable, \( \mathbf{rv} \), is a 2D vector. The scatter plot for this vector has a slope of \( \tan(\theta) = 0.41 \). The variances along the axes are 4 and 0.25. \( \mathbf{rv} \) is a graph of the principal axes which we will use for later comparison with simulations.

```plaintext
ndist = NormalDistribution[0,1];theta = Pi/8; bigvar = 4.0; smallvar = 0.25; alpha = N[Cos[theta]]; beta = N[Sin[theta]]; rv := 
{bigvar x1 alpha + smallvar y1 beta, 
brivar x1 beta - smallvar y1 alpha}/.{x1-> Random[ndist],y1-> Random[ndist]};
gprincipalaxes = Plot[{x beta, x (-1/beta)}, {x,-4,4},
PlotRange->{-4,4},{-4,4},
PlotStyle->{RGBColor[1,0,0]},
AspectRatio->1,DisplayFunction->Identity];
x1 and y1 are correlated. Let's view a scatterplot of samples from these two correlated Gaussian random variables.

npoints = 200;
rvsamples = Table[rv,{n,1,npoints}];
g1 = ListPlot[rvsamples,PlotRange->{-4,4},{-4,4},
AspectRatio->1,DisplayFunction->Identity];
Show[g1,gprincipalaxes, DisplayFunction-> $DisplayFunction];
```

Standard Principal Components Analysis (PCA)

Let \( E[\cdot] \) stand for the expected or average of a random variable. The covariance matrix of a of vector random variable, \( \mathbf{x} \), is: \( E[(\mathbf{x}-E[\mathbf{x}])(\mathbf{x}-E[\mathbf{x}])^T] \). Let's compute the autocovariance matrix for \( \mathbf{rv} \). The calculations are simpler because the average value of \( \mathbf{rv} \) is zero. As we would expect, the matrix is symmetric:
The variances of the two inputs (the diagonal elements) are due to the projections onto the horizontal and vertical axis of the generating random variable.

Now we will calculate the eigenvectors or the autocovariance matrix.

The eigenvalues give the ratio of the variances of the projections of the random variables \( \text{rv}[1] \) and \( \text{rv}[2] \) along the principal axes.

Note that the off-diagonal elements (the terms that measure the covariation of the transformed random variables) are zero. Further, because the variance of one of the projections is near zero, one can in fact dispense with this component and achieve a good approximate coding of the data with just one coordinate.
Neural networks and principal components

Neural network model using Hebb together with Oja’s rule for extracting the dominant principal component


Consider the following linear neural network. The input and output values are represented by vectors $x$ and $y$ respectively. The connection weights are represented by matrix $Q$.

$$
\begin{align*}
q_{11} & x_1 \\
q_{21} & x_2 \\
q_{12} & y_1 \\
q_{22} & y_2
\end{align*}
$$

We will combine the outer product form of Hebb’s rule, together with Oja’s modification. Without Oja’s rule, the Hebb rule does not place a limit on the size of the weights. Recall that Oja’s rule constrains the sum of the squares of the weights to approach 1. We will set the initial values of the weight matrix to random values between 0 and 1.

```
For[i=1,i<=npoints,i++,
  x = rv; y = Q.x;
  Q = Q + \alpha (Outer[Times,y,x] - Q y y);
  If[Mod[i,5]==0,
    pl = Join[pl,{{Q[1,2]/Q[1,1], Q[2,2]/Q[2,1] }}];
  ];
]
```

Let’s plot the slopes of projection axes as a function of iterations. We’ve sampled every 5th value, using `Mod[i,5]`, and stored it in `pl`.

```
ListPlot[Map[#[[2]]&,p1], AxesOrigin->{0,0}, PlotJoined->True,
  DisplayFunction->Identity,PlotStyle->{RGBColor[0,.5,0]};
ListPlot[Map[#[[1]]&,p1], AxesOrigin->{0,0}, PlotJoined->True,
  DisplayFunction->Identity,
  PlotStyle->{RGBColor[0,0,1]}];
Show[%,%%, DisplayFunction->$DisplayFunction];
```

There is some random fluctuation in the weights. We can obtain more stability by having a timeconstant over which the Hebbian term and the variance of $y$ are averaged.

```
For[i=1,i<=npoints,i++,
  x = rv; y = Q.x;
  Q = Q + \alpha (Outer[Times,y,x] - Q y y);
  If[Mod[i,5]==0,
    pl = Join[pl,{{Q[1,2]/Q[1,1], Q[2,2]/Q[2,1] }}];
  ];
]
```

We can see how well the coordinate transformation fits the principal axes of a sample scatter plot:

```
Show[%,%, DisplayFunction->$DisplayFunction];
```

Note that this different from standard matrix multiplication.
You can verify that the network does a good job of extracting the principal component. Recall that the slope for the population distribution is $\tan(\theta)$:

$\sqrt{2} = 0.414214$

The only problem with this network is that having two output neurons is redundant—they both pull out the same principal component—the dominant axis. The slopes for both are:

$p1[[\text{Length}[p1]]]$

{0.400705, 0.400705}

A generalization of Oja's rule for extracting all of the principal components with a "Neural network" (Sanger, 1989)


We will use the same network as in the above example. However, the learning rule will be asymmetric. The generalization of Oja's term is given by: $LT \text{Outer}[\times, y, x] \cdot Q$, where $LT$ is a lower triangular matrix. The entries above the diagonal are all zero, and the entries below and including the diagonal are one.

```plaintext
size = 2;
LT = Table[If[i>=j,1,0],{i,size},{j,size}];
npoints = 1200;
p1 = {}; \[alpha] = 0.1;
Q = Table[Random[]], {size}, {size}];
For[i=1,i<=npoints,i++,
x = rv; y = Q.x;
\delta Q = (Outer[\times, y, x] - (LT Outer[\times, y, y]).Q);
Q = Q + \alpha \delta Q;
If[\text{Mod}[i,1]==0,
p1 = Join[p1,{{Q[[1,2]]/Q[[1,1]], Q[[2,2]]/Q[[2,1]]}}]]];
```
PCA and natural images

Break a large image into a series of subimages.

The idea is that each subimage will be used as a statistical sample. We compute the outer product of each, and then average all 16 to get an estimate of the autocovariance matrix.

The more samples the better. So this works best with a larger image, such as GrayLonesome256x256.jpg or Graygranite256x256.jpg

```
<< Statistics`MultiDescriptiveStatistics`
```

Subtract off the mean.

```
subfacelist2 = Table[subfacelist[[i]] - Mean[subfacelist[[i]]], {i, 1, 256}];
```
Calculate the autocovariance matrix

```math
\text{temp} = \text{Table}[0.0, \{256\}, \{256\}];
\text{For}[i = 1, i < \text{Dimensions}[\text{subfacelist}][1], i++,
    \text{temp} = \text{N}[\text{Outer}[\text{Times}, \text{subfacelist}[[i]], \text{subfacelist}2[[i]]]] + \text{temp};]
```

```math
\text{ListDensityPlot}[\text{temp}, \text{Mesh} \rightarrow \text{False}];
```

Calculate the eigenvectors and eigenvalues of the autocovariance matrix

```math
\text{eigentemp} = \text{Eigenvectors}[\text{temp}];
\text{eigenvaluestemp} = \text{Eigenvalues}[\text{temp}];
\text{ListPlot}[\text{Chop}[	ext{eigenvaluestemp}]];\text{
```

Display the first 32 eigenvectors as "eigenpictures"

```math
\text{Table}[
    \text{ListDensityPlot}[\text{Partition}[	ext{eigentemp}[[i]], 16], \text{Mesh} \rightarrow \text{False}], \{i, 1, 32\}];
```

"
Efficient coding by V1

Olshausen & Field: Primary cortex

We might expect something like Fourier analysis of the image to result in efficient coding because of the close relationship between Fourier rotations and Karhunen-Loeve transformations (e.g. Appendix A, Andrews, 1983). Fourier coefficients for natural images tend to be uncorrelated. Some work has been completed toward a functional explanation for the orientation and spatial frequency tuning properties of cortical receptive fields based on the statistics of natural images (Field, 1987; Snyder), but the story is far from complete. Barlow has argued that a decorrelated representation of sensory information is important for efficient learning (Barlow, 1990).

There has been progress studying the relationship between self-organizing models of visual cortex, and efficient coding of image information. For more on this, see: Linsker, R. (1990) and Barlow, H. B., & Foldiak, P. (1989). Linsker's computational studies show, for example, that orientation tuning, and band-pass properties of simple cells can emerge as a consequence of maximum information transfer (in terms of variance) given the constraint that the inputs are already band-pass, and the receptive field connectivity is a priori limited.

We will see in the next lecture that cells in the visual cortex send their visual information to an incredibly complex, and yet structured collection of extra-striate areas. Any hypothesized function of striate cortex must eventually take into account what the information is to be used for. In the next lecture, we will give a quick overview of extra-striate visual cortex, and introduce the computational problem of estimating scene properties from image data.

In 1996, Olshausen and Field showed that one could derive a set of basis functions that have the same characteristics as the ensemble of visual simple cells in primary visual cortex by requiring two simple constraints:

1) One should be able to express the image as a weighted sum of the basis functions.
2) The total sum of activity across the ensemble should, on average, be small. This latter constraint is called "sparse coding". That is, a typical input image should activate a relatively small fraction of neurons in the ensemble.
Edge Detection

Introduction

The Noise/Scale trade-off

The definition of edge detection is very tricky—exactly what do we want to detect? We would like to label "significant" intensity changes in the image. One definition of significant edges is that they are the ones with the biggest change in intensity. The biggest would correspond to step changes. In Mathematica, these can be modeled as \( g(x) = \text{UnitStep}[x] \). One of the first problems we encounter is that edges in an image are typically fuzzy, either due to optical blur in the imaging device, or because the scene causes of edges are not changing abruptly. Our generative model for image data \( f \) is to convolve the step with a blur function:

\[
 f(x) = g(x - x') \text{blur}(x') \frac{d}{ds} g * \text{blur} \]

where \( g() \) is the signal to be detected or estimated. \( g(x) \) is a step function:

\[
 \sum_{x', y} \left[ I(x, y) - \sum_i a_i \phi_i(x, y) \right]^2 + \sum_i S' \]
Depending on the type of blur, the image intensity profile \( f(x) \) will look more or less like:

\[
\text{edge}[x_, s_] := \frac{1}{1 + \exp[-x/s]} \\
\text{Plot}[\text{edge}[x, .3], \{x, -2, 2\}, \text{Ticks} \to \text{None}];
\]

One way of locating the position of the edge in this image would be to take the first derivative of the intensity function, and then mark the edge location at the peak of the first derivative:

\[
\text{dedge}[u_, s_] := \frac{\text{D}[\text{edge}[x, t], x] \rvert_{x \to u}}{\text{.} \rvert_{t \to s}} \\
\text{Plot}[\text{dedge}[u, .3], \{u, -2, 2\}, \text{Ticks} \to \text{None}];
\]

Alternatively, we could take the second derivative, and look for zero-crossings to mark the edge location.

\[
\text{d2edge}[u_, s_] := \frac{\text{D}[\text{dedge}[x, t], x] \rvert_{x \to u}}{\text{.} \rvert_{t \to s}} \\
\text{Plot}[\text{d2edge}[u, .3], \{u, -2, 2\}, \text{Ticks} \to \text{None}];
\]

So far so good. But real images rarely have a nice smooth intensity gradation at points that we subjectively would identify as a clean edge. A more realistic generative model for intensity data would be:

\[
f(x) = \int g(x - x') \text{blur}(x') \, dx' + \text{noise}
\]

We'll add a fixed sample of high-frequency "noise":

\[
\text{noisyedge}[x_, s_] := \text{edge}[x, s] + 0.01 \cos[10 \, x] + 0.02 \sin[10 \, x] + 0.03 \cos[12 \, x] + 0.04 \sin[12 \, x] + -0.01 \cos[13 \, x] + -0.03 \sin[13 \, x] + 0.01 \cos[14 \, x] + 0.01 \sin[14 \, x] + -0.04 \cos[25 \, x] + -0.02 \sin[25 \, x] + 0.02 \cos[26 \, x] + 0.03 \sin[26 \, x]; \\
\text{Plot}[\text{noisyedge}[x, .3], \{x, -2, 2\}, \text{Ticks} \to \text{None}];
\]

Now, if we take the derivative, there are all sorts of peaks, and the biggest isn't even where the edge is:

\[
\text{dnoisyedge}[u_, s_] := \frac{\text{D}[\text{noisyedge}[x, t], x] \rvert_{x \to u}}{\text{.} \rvert_{t \to s}} \\
\text{Plot}[\text{dnoisyedge}[u, .3], \{u, -2, 2\}, \text{Ticks} \to \text{None}];
\]
Looking for zero-crossings looks hopeless:

\[
\Sigma = ((\sigma, 0), (0, \sigma)) \quad \mu = (0, 0)
\]

\[
\text{ndist} = \text{MultinormalDistribution}[\mu, \Sigma]
\]

\[
\text{PDF}[\text{ndist}, (x, y)]
\]

The order of the operators doesn't matter, so one can take the Laplacian of the Gaussian first, and then convolve this del-squared G kernel with the image, or one can blur the image first, and then take the second derivatives:

\[
r(x, y) = \nabla^2 (G_\sigma * g)
\]

As \(\sigma\) approaches zero, \(G_\sigma\) becomes a delta function, and the \(\nabla^2 G_\sigma\) becomes a Laplacian \(\nabla^2\), i.e., a second derivative operator. For small \(\sigma\), the detector is sensitive to noise. For large \(\sigma\), it is less sensitive to noise, but misses edges. The biological appeal of the Marr-Hildreth detector is that lateral inhibitory filters provide the \(\nabla^2 G_\sigma\) kernel. But what about the oriented filters in the cortex? One interpretation is in terms of 1st derivatives.

One could also build zero-crossing detectors by ANDing the outputs of appropriately aligned center-surround filters effectively building oriented filters (Marr and Hildreth).

**Edge detection using 1st derivatives**

Because of the orientation selectivity of cortical cells, they have sometimes been interpreted as edge detectors. We noted earlier how a sine-phase Gabor function filter (1 cycle wide) would respond well to an edge oriented with its receptive field. These sine-phase or odd-symmetric filters can also be viewed as spatial derivatives. How can we combine oriented filters to signal an edge?

The first-derivative operation takes the gradient of the image. From calculus, you learned that the gradient of a 2D function evaluated at \((x, y)\) is a vector that points in the direction of maximum change. So taking the gradient of an image should produce a vector field where the vectors are perpendicular to the edges. The length of the gradient is a measure of the steepness of the intensity gradient.

**The gradient of a function**

\[
\nabla g = \left( \frac{\partial g}{\partial x}, \frac{\partial g}{\partial y} \right)
\]

Let's try it with the cup image.

\[
\text{contcup} = \text{ListInterpolation}[\text{Transpose}[\text{cup}], \{(1, \text{width}), (1, \text{width})]\}]
\]
DensityPlot[contcup[x, y], {x, 1, 64}, {y, 1, 64}, Mesh -> False, PlotPoints -> 128];

Now let's put the x and y directions together and compute the squared gradient magnitude:

Now let's put the x and y directions together and compute the squared gradient magnitude:

Combining a smoothing pre-blur with 1rst derivatives

So the idea is to blur the image, take the first derivatives in the x and y directions, square each and add them up.

The x and y components of the gradient of the blur kernel:
Problems with interpreting V1 simple/complex cells as edge detectors

Although one can build edge detectors from oriented filters, simple cells cannot uniquely signal the presence of an edge for several reasons. One is that their response is a function of many different parameters. A low contrast bar at an optimal orientation will produce the same response as a bar of higher contrast at a non-optimal orientation. There is a similar trade-off with other parameters such as spatial frequency and temporal frequency. In order to make explicit the location of an edge from the responses of a population of cells, one would have to compute something like the "center-of-mass" over the population, where response rate takes the place of mass. Another problem is that edge detection has to take into account a range of spatial scales. We discussed evidence earlier that the cortical basis set does encompass a range of spatial scales, and in fact may be "self-similar" across these scales. See Koenderink (1990) for a theoretical discussion of "ideal" receptive field properties from the point of view of basis elements.

Segmentation & Why edge detection is hard

The finding of Knill & Kersten

- Land & McCann's "Two squares and a happening"

Combing signal detection theory with edge detection

Yuille and colleagues. Canny.
The left half looks lighter than the right half. But, let’s plot the intensity across a horizontal line:

\[
\text{new} = \text{picture; new}[[128, \text{All}]] = 0; \\
\text{ListDensityPlot[} \text{new, Mesh} = \text{False, PlotRange} \rightarrow \{0, 1\}; \\
\text{ListPlot[} \text{picture}[[128]], \text{PlotJoined} \rightarrow \text{True, PlotRange} \rightarrow \{0.2, .8\}]\]

The two ramps are identical...tho' not too surprising in that that is how we constructed the picture. How can we explain this illusion based on what we've learned so far?

One explanation is that the visual system takes a spatial derivative of the intensity profile. Recall from calculus that the second derivative of a linear function is zero. So a second derivative should filter out the slowly changing linear ramp in the illusory image. We approximate the second derivative with a discrete kernel (-1, 2, -1).

The steps are: 1) take the second derivative of the image; 2) threshold out small values; 3) integrate twice to undo the second derivative and "restore" the picture.

\[
\text{filter} = (-1, 2, -1); \\
\text{fspicture} = \text{ListConvolve[filter, picture[[128]]];} \\
\text{ListPlot[} \text{fspicture, PlotJoined} \rightarrow \text{True, PlotRange} \rightarrow \{-0.1, .1\}]\]

\[
\text{integratefspicture} = \text{FoldList[Plus, fspicture[[1]], fspicture];} \\
\text{integratefspicture2} = \text{FoldList[Plus, integratefspicture[[1]], integratefspicture];} \\
\text{ListPlot[} \text{integratefspicture2, PlotJoined} \rightarrow \text{True}]\]

To handle gradients that aren't perfectly linear, we could add a threshold function to set small values to zero before re-integrating:

\[
\text{threshold}[x_, y_] := \text{If}[x > y, x, 0]; \text{SetAttributes[threshold, Listable];} \\
\text{fspicture} = \text{threshold[fspicture, 0.025];} \\
\text{ListPlot[} \text{fspicture, PlotJoined} \rightarrow \text{True}]\]

Or one can take just the first derivative, followed by the threshold function.

Knill & Kersten’s "Two cylinders and no happening"

But is edge enhancement and spatial filtering a good way to explain the lightness effect? Up until the early 1990's many people thought so, and this was a standard textbook explanation of these kinds of lightness illusions.

What if we measure the intensity across a horizontal line in the "slab" on the left, and the "two-cylinders" on the right?
They are also the same! They would both look something like this:

But the perceived lightness contrast for the slabs is significantly stronger than it is for the two cylinders. A spatial convolution derivative model would predict the same for both. The spatial convolution operation won't work as an explanation!

**Edge classification: Some causes of edges are more important than others: task-dependence**

We've seen that uncertainty due to noise and spatial scale confound reliable edge detection. But there is another reason why edge detection is hard—local intensity gradients can have several possible meanings. So on the one hand, it still makes sense to interpret lateral inhibitory filters and oriented cortical filters as possible components of an edge detection system, we have to allow for considerable uncertainty in the significance of their outputs—i.e. a local edge detector typically has a low signal-to-noise ratio for a variety of ways of defining signals. Our goal is to take a look at an alternative view of early spatial filtering, as efficient encoding. But first, let's look at an even simpler example that doesn't even involve spatial interactions.

---

**Vision places a higher utility on surface and material edges than on other types. Surface edges are used differently from material edges**

**Natural images & segmentation**

- Lonesome peak

Where to draw the contours?
Next time

- Image-based modeling: geometry, contours & long lines
- Surfaces from images
- Scene-based modeling

Appendices

Morrone & Burr: polarity sensitive & polarity insensitive

- Morrone and Burr edge/bar detectors

  Suppose we convolve an input signal with an even filter (e.g. Gaussian enveloped cosine-wave) to produce response $R_e$, and then convolve the same input with an odd filter (say, a Gaussian enveloped sine-wave) to produce response $R_o$. The filters are orthogonal to each other, and so are the responses. $R_e$ will tend to peak at "bars" in the image whose size is near half the period of the cosine-wave. $R_o$ will tend to peak near edges.

  The local contrast "energy" is defined to be $\sqrt{R_e^2 + R_o^2}$. Morrone and Burr showed that the local energy peaks where the Fourier components of an image line up with zero-phase—i.e. at points where the various Fourier components are all in sine-phase. These points are edges. But it also peaks near bar features, arguably also interesting image features where the phase coherence is at 90 degrees. In addition to its neurophysiological appeal, a particularly attractive feature of this model is that if one adds up responses over multiple spatial scales, evidence accumulates for edges because the local energy peaks coincide there. They also showed how their model could be used to explain Mach bands.

Mach bands & the Morrone & Burr edge detector

```math
size = 256; Clear[y];
low = 0.2; hi = 0.8;
y[x_] := low /; x < size/3
y[x_] := ((hi - low)/(size/3)) x + (low - (hi - low)) /; x >= size/3 && x < 2*size/3
y[x_] := hi /; x > 2*size/3
Plot[y[x], {x, 0, 256}, PlotRange -> {0, 1}];
picture = Table[Table[y[i], {i, 1, size}], {i, 1, size}]; ListDensityPlot[picture, Frame -> False, Mesh -> False, PlotRange -> {0, 1}, AspectRatio -> Automatic];
```
Gabor filters

\[
\begin{align*}
\text{sgabor}[x_, y_, fx_, fy_, sig_] &= \text{N} \exp\left(-x^2 - y^2 \right) \left(2 \text{sig} \right) \sin\left(2 \pi \left(fx x + fy y \right)\right) \cdot \text{N} \exp\left(-x^2 - y^2 \right) \left(2 \text{sig} \right) \cos\left(2 \pi \left(fx x + fy y \right)\right) \\
\text{cgabor}[x_, y_, fx_, fy_, sig_] &= \text{N} \exp\left(-x^2 - y^2 \right) \left(2 \text{sig} \right) \sin\left(2 \pi \left(fx x + fy y \right)\right) \cdot \text{N} \exp\left(-x^2 - y^2 \right) \left(2 \text{sig} \right) \cos\left(2 \pi \left(fx x + fy y \right)\right)
\end{align*}
\]

\[
f_{\text{size}} = 16; \text{sfilter} = \text{Table}[\text{sgabor}\left[(i - \text{size}/2), (j - \text{size}/2), 0, 1/8, 4\right], (i, 0, \text{size}), (j, 0, \text{size})]; \text{sfilter} = \text{Chop}[\text{sfilter}]; \text{ListDensityPlot}[\text{sfilter}, \text{Mesh} \to \text{False}, \text{PlotRange} \to (-1, 1)];
\]

\[
f_{\text{size}} = 16; \text{cfilter} = \text{Table}[\text{cgabor}\left[(i - \text{size}/2), (j - \text{size}/2), 0, 1/8, 4\right], (i, 0, \text{size}), (j, 0, \text{size})]; \text{cfilter} = \text{Chop}[\text{cfilter}]; \text{ListDensityPlot}[\text{cfilter}, \text{Mesh} \to \text{False}, \text{PlotRange} \to (-1, 1)];
\]

Apply odd (sine) filter

\[
f_{\text{spicture}} = \text{ListConvolve}[\text{sfilter}, \text{picture}]; \text{ListDensityPlot}[f_{\text{spicture}}, \text{Mesh} \to \text{False}];
\]

Apply even (cosine) filter

\[
f_{\text{cpicture}} = \text{ListConvolve}[\text{cfilter}, \text{picture}]; \text{ListDensityPlot}[f_{\text{cpicture}}, \text{Mesh} \to \text{False}];
\]

Look for peaks in local contrast energy

\[
ss = \sqrt{f_{\text{spicture}}^2 + f_{\text{cpicture}}^2}.
\]
The Gibb's Sampler & Markov Random Field images

```math
size = 64;
brown = N[Table[128, {i, 1, size}, {i, 1, size}]];
```

```math
next[x_] := Mod[x, size] + 1;
previous[x_] := Mod[x - 2, size] + 1;

For[j = 1, j < size + 1, j++,
For[i = 1, i < size + 1, i++,
neighaverage = 0.25 * (brown[[next[i], j]] + brown[[i, next[j]]] + 
brown[[i, previous[j]]] + brown[[previous[i], j]]);
Clear[ndist];
ndist = NormalDistribution[neighaverage, .25];
brown[[i, j]] = Random[ndist];
];
]
```

```math
ListDensityPlot[brown, Mesh -> False];
```

```math
ListDensityPlot[ss, Mesh -> False];
```

```math
ListPlot[ss[[128]]];
```

```math
ListDensityPlot[brown, Mesh -> False];
```
histobrown = histogram[brown];
ListPlot[histobrown, PlotStyle -> PointSize[0.015], PlotRange -> {0, 0.001}];
entropy[histobrown]

0.767841

Geman's potential:

f[x_, n_] := Sqrt[Abs[x]^n / (1 + Abs[x]^n)];
Plot[f[x, 4], {x, -5, 5}]

References


